Reading Assignment

- This lecture: 3.4
- Next lecture: 3.4
Outline

Software Processor Modeling

Application Layer

Operating System Layer
Software Processor Modeling

- **General-purpose processors**
  - Processes are usually specified as sequential programs.
  - To estimate system design metrics, one has to consider not only the programs but the supporting software (e.g. OS).
  - Much more complicated than the above case. Will be our focus.

- **Challenges**
  - Most estimations of system design metrics, e.g. latency, throughput, and power consumption, depend on simulation.
  - Models enable fast simulations while provide accurate (relatively) estimations are desired.
  - At what level should we model software processors?
Considerations

▶ At instruction level, processors can be modeled as FSMs, and processes can be modeled as assembly programs. However,
  ▶ Simulations are slow as the supporting software are treated the same way.
  ▶ With all the details mixed together, it is difficult for the designers to make design decisions for improvements, especially when multiple processes are mapped to a single processor.

▶ Separate the computation into layers
  ▶ E.g. processor, OS/library, application processes
  ▶ Allow designers to reason about various parts of the system.
  ▶ Certain functionality, e.g. those from OS/library, can be simulated faster at levels higher than instructions.

▶ Software processor modeling is not limited only to the processors themselves but should include all pieces of the supporting software.
Considerations

- At instruction level, processors can be modeled as FSMs, and processes can be modeled as assembly programs. However,
  - Simulations are slow as the supporting software are treated the same way.
  - With all the details mixed together, it is difficult for the designers to make design decisions for improvements, especially when multiple processes are mapped to a single processor.
- Separate the computation into layers
  - E.g. processor, OS/library, application processes
  - Allow designers to reason about various parts of the system.
  - Certain functionality, e.g. those from OS/library, can be simulated faster at levels higher than instructions.
- Software processor modeling is not limited only to the processors themselves but should include all pieces of the supporting software.
Considerations

- At instruction level, processors can be modeled as FSMs, and processes can be modeled as assembly programs. However,
  - Simulations are slow as the supporting software are treated the same way.
  - With all the details mixed together, it is difficult for the designers to make design decisions for improvements, especially when multiple processes are mapped to a single processor.

- Separate the computation into layers
  - E.g. processor, OS/library, application processes
  - Allow designers to reason about various parts of the system.
  - Certain functionality, e.g. those from OS/library, can be simulated faster at levels higher than instructions.

- Software processor modeling is not limited only to the processors themselves but should include all pieces of the supporting software.
Typical Modeling of Software Processor

**FIGURE 3.8**  Processor modeling layers  
(Gajski et al.)
Typical Layers of Computation

- **Processor Hardware (HW)**
  - Actual hardware that communicates with the other parts of the system.

- **Hardware Abstraction Layer (HAL)**
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.

- **Operating System (OS)**
  - Provide necessary services like multi-tasking, scheduling, inter-process communication

- **Application**
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.
  - Notion of time can be introduced as execution delay.
  - Via back-annotation
Typical Layers of Computation

- Processor Hardware (HW)
  - Actual hardware that communicates with the other parts of the system.

- Hardware Abstraction Layer (HAL)
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.

- Operating System (OS)
  - Provide necessary services like multi-tasking, scheduling, inter-process communication

- Application
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.
  - Notion of time can be introduced as execution delay.
    - Via back-annotation
Typical Layers of Computation

- Processor Hardware (HW)
  - Actual hardware that communicates with the other parts of the system.

- Hardware Abstraction Layer (HAL)
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.

- Operating System (OS)
  - Provide necessary services like multi-tasking, scheduling, inter-process communication

- Application
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.
  - Notion of time can be introduced as execution delay.
  - Via back-annotation
Typical Layers of Computation

- **Processor Hardware (HW)**
  - Actual hardware that communicates with the other parts of the system.

- **Hardware Abstraction Layer (HAL)**
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.

- **Operating System (OS)**
  - Provide necessary services like multi-tasking, scheduling, inter-process communication

- **Application**
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.
  - Notion of time can be introduced as execution delay.
  - Via back-annotation
Typical Layers of Computation

- Processor Hardware (HW)
  - Actual hardware that communicates with the other parts of the system.

- Hardware Abstraction Layer (HAL)
  - Provide abstractions of actual hardware as canonical interfaces as most modern OS are hardware independent.

- Operating System (OS)
  - Provide necessary services like multi-tasking, scheduling, inter-process communication

- Application
  - With the help of OS, we can still model multiple processes mapped to the same processor as they are.

- Notion of time can be introduced as execution delay.
  - Via back-annotation
Outline

Software Processor Modeling

Application Layer

Operating System Layer
Applications are modeled as communicating processes.

- All this processes are running on the SAME processor so somewhat you may ignore the costs associated with their communications.

- Processes may be composed hierarchically, e.g. through the fork-join model.
  - Process scheduling is handled at the OS layer and is irrelevant at this layer, though it may affect the performance of the system.

- Communication mechanisms: shared variable and message passing
  - Message passing could be in its special and high-level forms like events, semaphores, queues, and abstract channels.
  - Message passing is most likely implemented at the OS layer, though we need to include the specification for applications.
Applications are modeled as communicating processes.

- All this processes are running on the SAME processor so somewhat you may ignore the costs associated with their communications.

Processes may be composed hierarchically, e.g. through the fork-join model.

- Process scheduling is handled at the OS layer and is irrelevant at this layer, though it may affect the performance of the system.

Communication mechanisms: shared variable and message passing

- Message passing could be in its special and high-level forms like events, semaphores, queues, and abstract channels.
- Message passing is most likely implemented at the OS layer, though we need to include the specification for applications.
Applications are modeled as communicating processes.

- All this processes are running on the SAME processor so somewhat you may ignore the costs associated with their communications.

Processes may be composed hierarchically, e.g. through the fork-join model.

- Process scheduling is handled at the OS layer and is irrelevant at this layer, though it may affect the performance of the system.

Communication mechanisms: shared variable and message passing

- Message passing could be in its special and high-level forms like events, semaphores, queues, and abstract channels.
- Message passing is most likely implemented at the OS layer, though we need to include the specification for applications.
Specifying Applications

- It is quite common that the initial application specifications are migrated from some other systems.
  - Computations are specified in a programming language like C/C++
  - Except shared variables, communications are specified as function calls to certain library/OS.
  - Process management are specified in similar ways like communication.
- Using a system design language like SystemC, the effort of migrating code could be minimized.
  - Computations are included directly.
  - Function calls for communications/process management can be replaced by corresponding primitives.
Specifying Applications

- It is quite common that the initial application specifications are migrated from some other systems.
  - Computations are specified in a programming language like C/C++
  - Except shared variables, communications are specified as function calls to certain library/OS.
  - Process management are specified in similar ways like communication.
- Using a system design language like SystemC, the effort of migrating code could be minimized.
  - Computations are included directly.
  - Function calls for communications/process management can be replaced by corresponding primitives.
Considerations for Simulation

- **Fast simulation**
  - Computation could be emulated in host instructions, usually much faster than simulating target instructions.
  - Simulation kernel may be optimized to reduce process management/scheduling overhead and communication costs, and to utilize the multi-processing power of the host.

- **Accurate estimations**
  - Back-annotation is necessary if timing behavior is of concern as simulation is not performed on targeted platform.
  - System design languages support back-annotations via execution timings, which are independent of their running times on host.
Considerations for Simulation

▶ Fast simulation
  ▶ Computation could be emulated in host instructions, usually much faster than simulating target instructions.
  ▶ Simulation kernel may be optimized to reduce process management/scheduling overhead and communication costs, and to utilize the multi-processing power of the host.

▶ Accurate estimations
  ▶ Back-annotation is necessary if timing behavior is of concern as simulation is not performed on targeted platform.
  ▶ System design languages support back-annotations via execution timings, which are independent of their running times on host.
Application Layer Modeling Example

FIGURE 3.9 Application layer

(Gajski et al.)
More about Back-Annotation

- Back-annotations may be included at various levels of computations.
  - E.g. the whole process or basic blocks
- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.
- Associating delay numbers with basic blocks will be able to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.
- There is always a trade-off between simulation speed and accuracy.
- The delay numbers may be obtained by estimation or measurement.
More about Back-Annotation

- Back-annotations may be included at various levels of computations.
  - E.g. the whole process or basic blocks

- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.

- Associating delay numbers with basic blocks will be able to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.

- There is always a trade-off between simulation speed and accuracy.

- The delay numbers may be obtained by estimation or measurement.
More about Back-Annotation

- Back-annotations may be included at various levels of computations.
  - E.g. the whole process or basic blocks
- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.
- Associating delay numbers with basic blocks will be able to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.
- There is always a trade-off between simulation speed and accuracy.
- The delay numbers may be obtained by estimation or measurement.
More about Back-Annotation

- Back-annotations may be included at various levels of computations.
  - E.g. the whole process or basic blocks

- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.

- Associating delay numbers with basic blocks will be able to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.

- There is always a trade-off between simulation speed and accuracy.
  - The delay numbers may be obtained by estimation or measurement.
More about Back-Annotation

- Back-annotations may be included at various level of computations.
  - E.g. the whole process or basic blocks
- As a process may react differently for different inputs, assigning it a simple delay number could be inaccurate.
- Associating delay numbers with basic blocks will be able to capture such dynamic behavior and thus improve accuracy.
  - However, it will slow down simulation because of the excessive interactions with the simulation kernel required by back-annotation.
- There is always a trade-off between simulation speed and accuracy.
- The delay numbers may be obtained by estimation or measurement.
Outline

Software Processor Modeling

Application Layer

Operating System Layer
We assume processes are running truly *concurrently* at application layer.

However, when mapped to a single processor, that’s not possible.

OS provides the illusion that processes are running concurrently,

- though actually they run sequentially on the processor.

The major focus of the OS layer is thus to model the scheduling of parallel processes on the sequential processor.
We assume processes are running truly *concurrently* at application layer.

However, when mapped to a single processor, that’s not possible.

OS provides the illusion that processes are running concurrently,
- though actually they run sequentially on the processor.

The major focus of the OS layer is thus to model the scheduling of parallel processes on the sequential processor.
We assume processes are running truly *concurrently* at application layer.

However, when mapped to a single processor, that’s not possible.

OS provides the illusion that processes are running concurrently,

- though actually they run sequentially on the processor.

The major focus of the OS layer is thus to model the scheduling of parallel processes on the sequential processor.
We assume processes are running truly *concurrently* at application layer.

However, when mapped to a single processor, that’s not possible.

OS provides the illusion that processes are running concurrently,

- though actually they run sequentially on the processor.

The major focus of the OS layer is thus to model the scheduling of parallel processes on the sequential processor.
Static vs. Dynamic Scheduling

- Static scheduling combines processes into a single task following a pre-defined static schedule.
  - E.g. you may interleave the statements of multiple processes to achieve a static round-robin scheduling.
  - Modeling of static scheduling is no harder than modeling of a sequential program.

- What if a process need to communicate for either data or synchronization and have to wait?
  - The task has to wait.
  - All the processes in the task halt.
  - May hurt system latency/throughput or even lead to artificial deadlocks.

- Dynamic scheduling allows OS, at runtime, to hang tasks that have to wait and to execute tasks that are ready to go.
  - For embedded systems, typically a Real-Time Operating System (RTOS) is used.
  - And thus a model of RTOS should be established.
  - Real-time here doesn’t imply low computing latency, but means mechanism to meet certain deadlines.
Static vs. Dynamic Scheduling

_static scheduling combines processes into a single task following a pre-defined static schedule._

▶ E.g. you may interleave the statements of multiple processes to achieve a static round-robin scheduling.
▶ Modeling of static scheduling is no harder than modeling of a sequential program.
▶ What if a process need to communicate for either data or synchronization and have to wait?
  ▶ The task has to wait.
  ▶ All the processes in the task halt.
  ▶ May hurt system latency/throughput or even lead to artificial deadlocks

Dynamic scheduling allows OS, at runtime, to hang tasks that have to wait and to execute tasks that are ready to go.

▶ For embedded systems, typically a Real-Time Operating System (RTOS) is used.
▶ And thus a model of RTOS should be established.
▶ Real-time here doesn’t imply low computing latency, but means mechanism to meet certain deadlines.
Static vs. Dynamic Scheduling

- Static scheduling combines processes into a single task following a pre-defined static schedule.
  - E.g., you may interleave the statements of multiple processes to achieve a static round-robin scheduling.
  - Modeling of static scheduling is no harder than modeling of a sequential program.
- What if a process needs to communicate for either data or synchronization and have to wait?
  - The task has to wait.
  - All the processes in the task halt.
  - May hurt system latency/throughput or even lead to artificial deadlocks
- Dynamic scheduling allows OS, at runtime, to hang tasks that have to wait and to execute tasks that are ready to go.
  - For embedded systems, typically a Real-Time Operating System (RTOS) is used.
  - And thus a model of RTOS should be established.
  - Real-time here doesn’t imply low computing latency, but means mechanism to meet certain deadlines.
Other OS Layer Modelings

- Process/task management
  - Synchronization
  - Timing, i.e. delay modeling
  - Realize communication channels as IPC primitives
    - Synchronization may be required.
Other OS Layer Modelings

- Process/task management
- Synchronization
  - Timing, i.e. delay modeling
  - Realize communication channels as IPC primitives
    - Synchronization may be required.
Other OS Layer Modelings

- Process/task management
- Synchronization
- Timing, i.e. delay modeling
  - Realize communication channels as IPC primitives
    - Synchronization may be required.
Other OS Layer Modelings

- Process/task management
- Synchronization
- Timing, i.e. delay modeling
- Realize communication channels as IPC primitives
  - Synchronization may be required.
Interaction between Application and OS Layers

*FIGURE 3.10* Operating system layer

(Gajski et al.)
RTOS Modeling Choices

For specification (a), tasks execute directly on the simulation kernel and scheduling is omitted.

There are two choices to model the actual RTOS scheduling algorithm.

(c) simulation at instruction level

(b) provide a transaction-level RTOS model

FIGURE 3.11 Operating system modeling

(Gajski et al.)
For specification (a), tasks execute directly on the simulation kernel and scheduling is omitted.

There are two choices to model the actual RTOS scheduling algorithm.

- (c) simulation at instruction level
- (b) provide a transaction-level RTOS model
Instruction vs. Transaction-Level RTOS Modelings

- **Instruction level simulation**
  - Need to compile all codes, including applications, to target instructions.
  - Simulation is done in an instruction-set simulator (ISS).
  - Very accurate, but slow

- Transaction-level RTOS models combine fast application simulation in host instructions and instruction-level modeling accuracy.
  - Use an abstract RTOS model that removes unnecessary implementation details
  - Only care about multi-tasking, preemption, interrupt handling, inter-process communication, and synchronization
  - Add only a negligible simulation overhead
  - Allow designers to consider important OS effects early on in the design process
Instruction vs. Transaction-Level RTOS Modelings

- **Instruction level simulation**
  - Need to compile all codes, including applications, to target instructions.
  - Simulation is done in an instruction-set simulator (ISS).
  - Very accurate, but slow

- **Transaction-level RTOS models combine fast application simulation in host instructions and instruction-level modeling accuracy.**
  - Use an abstract RTOS model that removes unnecessary implementation details
  - Only care about multi-tasking, preemption, interrupt handling, inter-process communication, and synchronization
  - Add only a negligible simulation overhead
  - Allow designers to consider important OS effects early on in the design process
Example for Scheduling and Other OS Services

![Diagram of task scheduling](image)

**FIGURE 3.12 Task scheduling**

(Gajski et al.)
Summary

- Processor modeling should include all pieces of supporting software, e.g. HAL and OS.
- Application layer includes functional models that can be efficiently simulated.
- Process/task scheduling are the major focus of OS layer modeling.
- Back-annotations enable accurate estimations.